# Zephyr RTOS: Creating modules and debugging applications in ModusToolbox™

## About this document

### Scope and purpose

This application note provides a detailed guide for developers interested in creating a custom module in Zephyr RTOS, an open-source real-time operating system (RTOS) for resource-constrained systems, and built with security in mind. It provides step-by-step instructions to integrate ModusToolbox™ libraries into a Zephyr module and seamlessly integrate them into the Windows-based development build system.

It also includes a tutorial on creating a littlefs sample application, from configuration to functionality verification, and expert guidance on debugging using Eclipse IDE for ModusToolbox™ to identify and troubleshoot issues in the Zephyr RTOS based application.

### Intended audience

This document is intended for anyone who needs to create a module for Zephyr RTOS environment using libraries of ModusToolbox™.

Table of contents

## Table of contents

# 1 Introduction

The Zephyr OS is based on a small-footprint kernel designed for use on resource-constrained systems from simple embedded environmental sensors and LED wearables to sophisticated smart watches and IoT wireless gateways. The Zephyr kernel supports multiple architectures. This application note uses the PSoC™ 6 MCU as the target platform. For more details on Zephyr, read the Introduction to Zephyr.

The following are the features enabled for PSoC™ 6 MCU in Zephyr:

- PSoC™ 6 Wi-Fi Bluetooth® Prototyping Kit integrated under *boards/arm* in Zephyr GitHub
- System clocks enabled on the board
- Simple tick source provided (no deep sleep support by default)
- Pin mapping for the kit's buttons, user LEDs, etc.
- Serial output mapped to the KitProg3 UART
- MPU and stack protection enabled
- Flashing and debugging is enabled via the `west` command

For more information on this platform, see board documentation.

## 1.1 PSoC™ 6 MCU integration hierarchy in Zephyr

PSoC™ 6 MCU is an ultra-low-power PSoC™ device with a dual-core architecture tailored for smart homes, IoT gateways, etc. The ModusToolbox™ software environment supports PSoC™ 6 MCU application development with a set of tools for configuring the device, setting up peripherals, and complementing your projects with world-class middleware.

The following figure illustrates the PSoC™ 6 MCU integration in Zephyr environment.

### Introduction

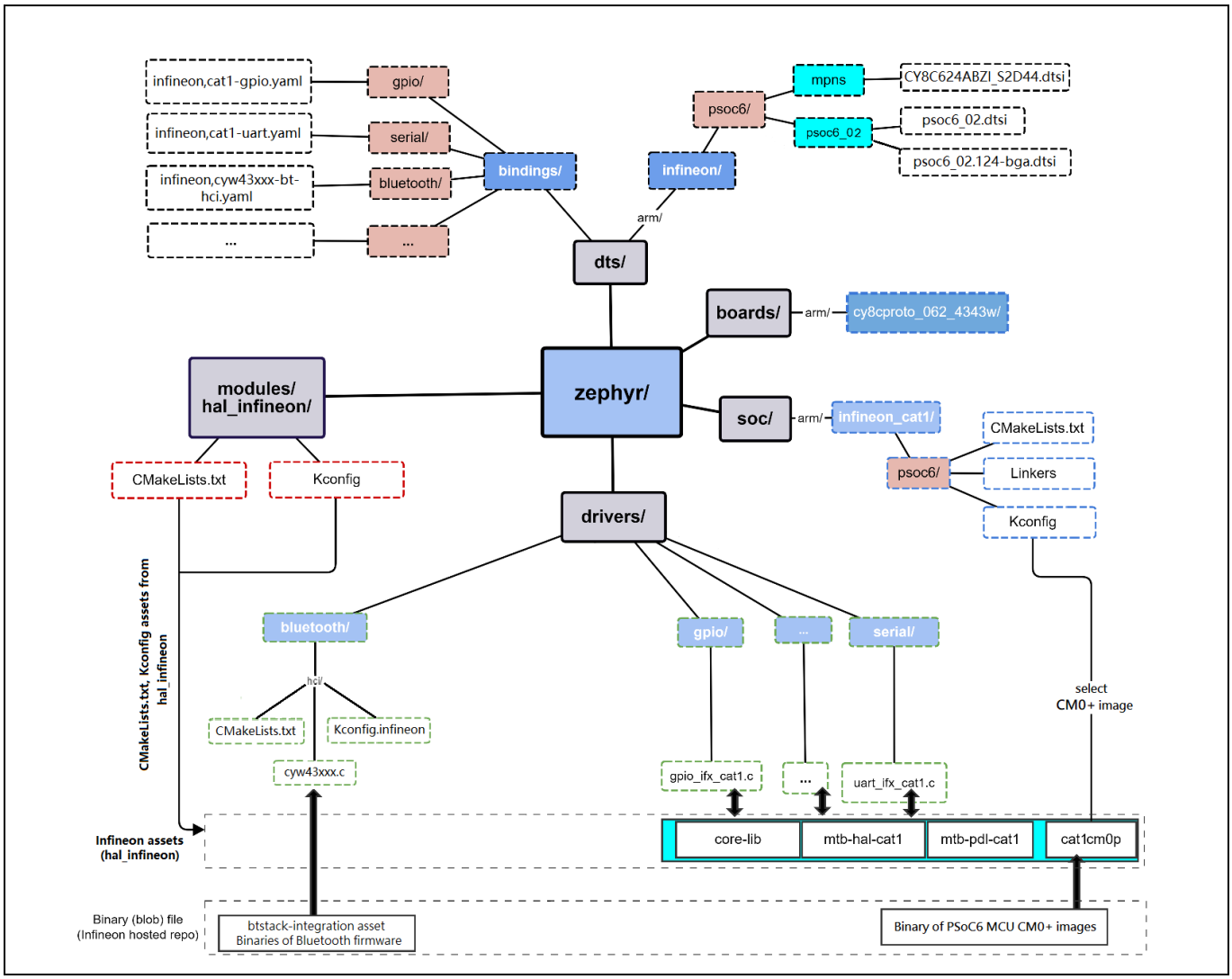


**Figure 1 PSoC™ 6 MCU integration hierarchy in Zephyr**

For more information on PSoC™ 6 MCU and ModusToolbox™ software, see Getting started with PSoC™ 6 MCU on ModusToolbox™ software.

## 1.2 Zephyr modules

Zephyr relies on the source code of several externally maintained projects in order to reuse as much well-established, mature code as possible. In the context of Zephyr's build system those are called "modules". These modules must be integrated with the Zephyr build system. An external project is required to have its own lifecycle outside the Zephyr project, that is, reside in its own repository, and have its own contribution and maintenance workflow and release process. Zephyr modules should not contain code that is written exclusively for Zephyr. Instead, such code should be contributed to the main Zephyr tree.

For more information about module repositories, module YAML file description, maintaining the codebase, synchronizing with upstream, integrating the modules in the Zephyr build system, see Zephyr documentation on modules.

This document demonstrates importing the mtb-littlefs and serial-flash libraries from ModusToolbox™ to create modules as an example and use them with the littlefs library available in Zephyr modules.

# 2 Development ecosystem

## 2.1 Hardware requirements

The PSoC™ 6 Wi-Fi Bluetooth® Prototyping Kit (CY8CPRORO-062-4343W) is a low-cost hardware platform that enables design and debug of PSoC™ 6 MCUs. It comes with a CY8CMOD-062-4343W module, industry-leading CAPSENSE™ for touch buttons and slider, onboard debugger/programmer with KitProg3, microSD card interface, 512-MB Quad SPI NOR flash, a PDM microphone, and a thermistor. It also includes a Murata LBEE5KL1DX module, based on the CYW4343W combo device.

The following hardware peripheral drivers of PSoC™ 6 MCU are integrated in the Zephyr environment:

- GPIO
- Serial (UART)

For more information on this platform, see board documentation.

## 2.2 Software requirements

Follow the instructions mentioned in the Getting started guide on Zephyr to install the required dependencies. See 2.4 to install the Python dependencies.

Install build tools for the required targets for Windows system including CMake, Python v3.8.x, DeviceTree Compiler (DTC), OpenOCD, and west build tools. These are required for Zephyr to compile, flash, and debug user applications.

Table 1 shows the setup flow tested on Windows 10 system for the PSoC™ 6 MCU platform.

**Table 1 Software tools used in this example**

| Tools | Version | Description |
|---|---|---|
| CMake | 3.20.0 or greater | Cross-platform build system |
| Python | 3.8.0 or greater | Programming language. Tested with version 3.8.10 |
| DeviceTree Compiler (DTC) | 1.4.6 or greater | Used for converting Device Tree Source (DTS) files to Device Tree Blob (DTB) binary files used by the kernel. |
| west | 1.0.0 or greater | Command-line tool for Zephyr that streamlines building, configuring, and debugging embedded systems across multiple platforms and development environments. It can be installed via pip3. |
| GNU Arm® toolchain | 10.3-2021.10 | Set of tools for developing embedded software on Arm®-based devices |
| CYPRESS™ Programmer | 4.0.0 or greater | Stand-alone, cross-platform, flash programmer tool that provides a graphical user interface to program, erase, verify, and read the flash of the target device. It supports HEX, SREC, ELF, and BIN programming file formats. Use CYPRESS™ Programmer as an alternate tool to flash the application into the board. |
| Cypress™ OpenOCD | 4.3.0 or greater | Free software that provides debugging, in-system programming, and boundary-scan testing. Cypress™ OpenOCD supports the following:<br>PSoC™ 61/62/63/64 product lines and the corresponding kits<br>KitProg3 onboard programmer<br>MiniProg4 standalone programmer |

## 2.3 Zephyr environment setup

On Windows, Zephyr requires environment variables for the build system type and path, as shown in the environment variables.

*Note:* *The environment variable information is cached during a build. If you are changing or experimenting with these values, delete the build directory from <HOMEPATH>/zephyrproject/zephyr path. Otherwise, the west compiler will continue to pick up the cached values for these environment variables.*

Set up the following environment variables of the installed build tools.

**Table 2 Environment variables**

| Variable | Value |
|---|---|
| Path | *C:\Users\<user_name>\AppData\Local\Programs\Python\Python38\Scripts*<br>*C:\Users\<user_name>\AppData\Local\Programs\Python\Python38*<br>*C:\Users\<user_name>\ModusToolbox\tools_2.4\openocd\bin*<br>*C:\Program Files\CMake\bin* |
| GNUARMEMB_TOOLCHAIN_PATH | *C:\Users\<user_name>\ModusToolbox\tools_2.4\gcc* |
| ZEPHYR_TOOLCHAIN_VARIANT | Gnuarmemb |

## 2.4 Get Zephyr and install Python dependencies

The following steps will guide you to create a new west workspace named “zephyrproject” in the desired location (Here, it is *C:*).

1. Install west:

```
pip3 install -U west
```

2. Get the Zephyr source code:

```
west init zephyrproject
cd C:\zephyrproject
west update
west blobs fetch hal_infineon
```

3. Export a Zephyr CMake package. This allows CMake to automatically load boilerplate code required for building Zephyr applications.

```
west zephyr-export
```

4. Install additional Python dependencies with pip3. Zephyr’s scripts\requirements.txt file declares these.

```
pip3 install -r C:\zephyrproject\zephyr\scripts\requirements.txt
```

## 2.5 Install Zephyr SDK

See Install Zephyr SDK.

# 3 Creating your first Zephyr application for PSoC™ 6

This section describes how to create a FreeRTOS application in ModusToolbox™ and then create similar applications in the Zephyr, using the littlefs filesystem (v1.0.0) application as an example.

The chapter is divided in the following parts:

- Create a FreeRTOS application for PSoC™ 6 MCU in ModusToolbox™
- Create the Zephyr application from your existing FreeRTOS application

## 3.1 Create a FreeRTOS application in ModusToolbox™ 2.4

Use the following instructions to create an application:

1. Download and install ModusToolbox™ software 2.4.

*Note:* *You may need add the following environment variable if you have multiple versions of ModusToolbox™ installed already on your Windows system.*

```
CY_TOOLS_PATHS = C:/Users/<USERNAME>/ModusToolbox/tools_2.4/
```

2. Create a new workspace (<ModusToolbox_workspace>) for the application using ModusToolbox™ 2.4.

Create the 'littlefs_filesystem' application in this workspace using instructions mentioned in the README.md file of the example.

## 3.2 Create a Zephyr application

To create an application in Zephyr, see the Application Development Guide from Zephyr. Alternatively, you can do the following:

### 3.2.1 Select the workspace and application directory structure

1. Create an application directory by using cmd.exe or manually in the Zephyr workspace ('zephyrproject'):

```
cd C:\zephyrproject
mkdir zephyr_example_filesystem_littlefs
```

*Note:* *Do not use spaces anywhere in the any directory/file path. It may result in error.*

2. Create the source code subdirectory named "src" under the application directory:

```
cd zephyr_example_filesystem_littlefs
mkdir src
```

3. Copy and paste your ModusToolbox™ application source code in the *src* subdirectory of Zephyr.

*Note:* *The application source code of ModusToolbox™ project is available in the following path:*

*<ModusToolbox_workspace_path>\Littlefs_Filesystem*

This example uses only the *main.c* file as source.

**Creating your first Zephyr application for PSoC™ 6**

4. Create another application subdirectory named “GeneratedSource” under the application directory:

```
mkdir GeneratedSource
```

5. Copy the following *GeneratedSource* files from the ModusToolbox™ application and paste into the Zephyr application’s *GeneratedSource* subdirectory.
   - *cycfg_pins.c*
   - *cycfg_pins.h*
   - *cycfg_qspi_memslot.c*
   - *cycfg_qspi_memslot.h*

   These files are available in the following path:

   *<ModusToolbox_workspace_path>\Littlefs_Filesystem\libs\TARGET_CY8CPROTO-062-4343W\COMPONENT_BSP_DESIGN_MODUS\GeneratedSource*

6. Create the *CMakeLists.txt* and *Prj.conf* files. See the Application Development Guide from Zephyr for more details.

The application directory structure should look similar to the following:

```
<Application>
Src
- main.c
GeneratedSource
- cycfg_pins.c/.h
- cycfg_qspi_memslot.c/.h
CMakeLists.txt
Prj.conf
```

## 3.2.2 View/modify/create modules

Do the following to add mtb-littlefs and serial-flash libraries as modules into the Zephyr build system.

1. Create and add the mtb-littlefs.yaml and serial-flash.yaml files in the C:\zephyrproject\zephyr\submanifests directory.

**Code Listing 1** ***mtb-littlefs.yaml***

```
manifest:
  projects:
    - name: infineon_mtb_littlefs
      url: https://github.com/cypresssemiconductorco/mtb-littlefs
      submodules: true
      revision: release-v2.0.0
      path: modules/lib/mtb-littlefs
      groups:
        - lib
```

### Creating your first Zephyr application for PSoC™ 6

**Code Listing 2** ***serial-flash.yaml***

```
manifest:
  projects:
    - name: infineon_serial_flash
      url: https://github.com/cypresssemiconductorco/serial-flash
      submodules: true
      revision: release-v1.3.0
      path: modules/lib/serial-flash
      groups:
        - lib
```

*Note:* *If you want to add any other libraries of ModusToolbox™ as modules in Zephyr, create other YAML files with the library name. See the `west manifest` to learn how to modify YAML files.*

2. After adding module YAML files in the *submanifests* directory, update the Zephyr modules by executing the following command:

```
west update
```

   Use the `west update` command to update the existing modules in Zephyr and add the new mtb-littlefs and serial-flash libraries in the following *modules* directories.

   *C:\zephyrproject\modules\lib\mtb-littlefs*

   *C:\zephyrproject\modules\lib\serial-flash*

*Note:* *The littlefs filesystem sample application in this example uses mtb-littlefs, serial-flash, and littlefs libraries. The littlefs library is already present in the Zephyr modules directory (C:\zephyrproject\modules\fs\littlefs); therefore, it is not added separately as a module.*

3. Once you create a YAML file for your library, update the *CMakeLists.txt* file for adding necessary includes and sources path. See Step 4 of Section 3.2.3.
4. Navigate to the path of littlefs library (*C:\zephyrproject\modules\fs\littlefs)* and execute the following command:

```
git checkout 9c7e232
```

*Note:* *This command fetches littlefs library v2.4.2. The littlefs filesystem application has been tested with this version.*

### 3.2.3 Modify Zephyr application source files

1. Open *cycfg_pins.h* and remove/comment the following lines:

```
#include "cycfg_notices.h"  //line number 33
#include "cycfg_routing.h"  //line number 38
```

2. Add the following macros in *cycfg_pins.h* in the beginning of the macros:

```
#define ioss_0_port_6_pin_4_HSIOM P6_4_CPUSS_SWJ_SWO_TDO
#define ioss_0_port_6_pin_6_HSIOM P6_6_CPUSS_SWJ_SWDIO_TMS
#define ioss_0_port_6_pin_7_HSIOM P6_7_CPUSS_SWJ_SWCLK_TCLK
```

### 3.2.4 Modify the *main.c* file

*Note: Based on the application being ported, there may be additional changes required.*

1. Make the following changes in the `Header files` section:

- Remove/comment the following header files:

```
#include "cy_pdl.h"
#include "cybsp.h"
#include "cy_retarget_io.h"
#include "FreeRTOS.h"
#include "task.h"
```

- Add the following header files:

```
#include "cycfg_pins.h"
#include <zephyr/kernel.h>
#include <zephyr/device.h>
```

2. Make the following changes in the macros section:

- Change the stack size for the thread as required. In this example, use 4X of the FreeRTOS task size is used.

```
#define LITTLEFS_TASK_STACK_SIZE            (2048U)
```

- Set the task priority. Define the `LITTLEFS_TASK_PRIORITY` as '6'. This CE originally used the FreeRTOS configuration provided task priority, thus need to be defined here.

```
#define LITTLEFS_TASK_PRIORITY              (6U)
```

### Creating your first Zephyr application for PSoC™ 6

- Define the button status. This is required because the *cybsp_types.h* file which has this definition is not included.

```
#define CYBSP_BTN_OFF                       (1U)
```

- Remove the following

```
#if (STORAGE_DEVICE_SD_CARD) && \
    (defined (TARGET_CY8CKIT_062_BLE) || \
    defined (TARGET_CY8CKIT_062_WIFI_BT) || \
    defined (TARGET_CY8CPROTO_062S3_4343W) || \
    defined (TARGET_CYW9P62S1_43438EVB_01))
#error The selected target does not support SD card.
#endif /* #if STORAGE_DEVICE_SD_CARD */
```

3. Make the following changes in the **function prototypes** section:

- Declare the `littlefs_task()` function as follows:

```
static void littlefs_task(void *dummy1, void *dummy2, void *dummy3);
```

4. Make the following changes in the **Global variables** section:

- Remove the following variables related to FreeRTOS:

```
static TaskHandle_t littlefs_task_handle;
volatile int uxTopUsedPriority;
```

- Declare the following. This is needed because of the difference in the HAL and PDL versions between the CE and the Zephyr workspace.

```
K_SEM_DEFINE(btn_isr_sema, 0, 1);
K_THREAD_DEFINE(littlefs_task_id, LITTLEFS_TASK_STACK_SIZE, littlefs_task,NULL, NULL, NULL,
LITTLEFS_TASK_PRIORITY, 0, -1);
cyhal_gpio_callback_data_t gpio_btn_callback_data;
```

5. Replace all the `printf()` statements with `printk()`.
6. Modify the `user_button_interrupt_handler()` function as follows. This is needed because of the difference in the HAL and PDL versions between the CE and the Zephyr workspace.

```
static void user_button_interrupt_handler(void *handler_arg, cyhal_gpio_event_t event)
{
    ARG_UNUSED(handler_arg);
    ARG_UNUSED(event);
    /* notify thread that data is available */
    k_sem_give(&btn_isr_sema);
}
```

### Creating your first Zephyr application for PSoC™ 6

7. Modify the `littlefs_task()` function as follows:

```
static void littlefs_task(void *dummy1, void *dummy2, void *dummy3)
```

- Add the following after `struct lfs_config lfs_cfg;` to avoid the 'unused variable' warning:

```
ARG_UNUSED(dummy1);
ARG_UNUSED(dummy2);
ARG_UNUSED(dummy3);
```

- Replace the FreeRTOS APIs with Zephyr APIs as follows:

Replace the following snippet:

```
while (true)
{
    if(1lu == ulTaskNotifyTake(pdTRUE, portMAX_DELAY))
    {
        /* Debounce the button press. */
        vTaskDelay(pdMS_TO_TICKS(DEBOUNCE_DELAY_MS));
        if(!cyhal_gpio_read(CYBSP_USER_BTN)) { break; }
    }
}
```

With the following snippet:

```
while (true)
{
    if(k_sem_take(&btn_isr_sema, K_MSEC(DEBOUNCE_DELAY_MS))!= 0)
    {
        /* Debounce the button press. */
        k_msleep(DEBOUNCE_DELAY_MS);
        if(!cyhal_gpio_read(CYBSP_USER_BTN)) { break; }
    }
}
```

8. Make the following changes in `main()` function.

- Modify the return type of `main()` function:

```
void main(void)
```

- Remove the functions related to initializing the BSP and the retarget-io library. The board initialization and the UART print message functionality are enabled by default in the Zephyr port for the PSoC™ 6 MCU support package.

```
/* Initialize the device and board peripherals */
result = cybsp_init() ;
CY_ASSERT (result == CY_RSLT_SUCCESS);
```

### Creating your first Zephyr application for PSoC™ 6

```
/* Initialize retarget-io to use the debug UART port */
result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
CY_RETARGET_IO_BAUDRATE);
CY_ASSERT (result == CY_RSLT_SUCCESS);
```

- Remove `uxTopUsedPriority` related to FreeRTOS-based debug awareness.
- Initialize the pins by calling the corresponding function at the beginning of `main()`:

```
init_cycfg_pins();
```

- Configure the user button interrupt. This is required because of the difference in the HAL and PDL versions between the CE and the Zephyr workspace.

```
/* Configure & the user button interrupt */
gpio_btn_callback_data.callback = user_button_interrupt_handler;
cyhal_gpio_register_callback(CYBSP_USER_BTN, &gpio_btn_callback_data);
```

- Replace the FreeRTOS APIs with the Zephyr APIs. Replace the following snippet:

```
xTaskCreate(littlefs_task, "Littlefs Task", LITTLEFS_TASK_STACK_SIZE,NULL,
(configMAX_PRIORITIES - 1), &littlefs_task_handle);


/* Start the RTOS scheduler. This function should never return */
vTaskStartScheduler();


(void) result; /* To avoid compiler warning */
for (;;)
{
}
```

With this snippet:

```
k_thread_start(littlefs_task_id);
```

See the API documentation for Zephyr kernel services for more details on Zephyr API.

The *main.c* file should look like the following:

```
#include "cyhal.h"
#include "cycfg_pins.h"
#include "lfs.h"
#include "lfs_sd_bd.h"
#include "lfs_spi_flash_bd.h"
#include <zephyr/kernel.h>
#include <zephyr/device.h>

/* When set to 1, SD card will be used instead of the NOR flash */
#define STORAGE_DEVICE_SD_CARD              (0)
```

### Creating your first Zephyr application for PSoC™ 6

```
#define LITTLEFS_TASK_STACK_SIZE            (2048U)
#define LITTLEFS_TASK_PRIORITY              (6U)
#define USER_BUTTON_INTERRUPT_PRIORITY      (7U)

/* Debounce delay for the user button. */
#define DEBOUNCE_DELAY_MS                   (50U)

/* Copied BSP defines */
#define CYBSP_BTN_OFF                       (1U)

static void littlefs_task(void *dummy1, void *dummy2, void *dummy3);

K_SEM_DEFINE(btn_isr_sema, 0, 1);
K_THREAD_DEFINE(littlefs_task_id, LITTLEFS_TASK_STACK_SIZE, littlefs_task,
            NULL, NULL, NULL, LITTLEFS_TASK_PRIORITY, 0, -1);
cyhal_gpio_callback_data_t gpio_btn_callback_data;

static void check_status(char *message, uint32_t status)
{
    if (0u != status)
    {
        printk("\n=========================================================\n");
        printk("\nFAIL: %s\n", message);
        printk("Error Code: 0x%08"PRIx32"\n", status);
        printk("\n=========================================================\n");

        while(true);
    }
}

static void print_block_device_parameters(struct lfs_config *lfs_cfg)
{
    printk("Number of blocks: %"PRIu32"\n", lfs_cfg->block_count);
    printk("Erase block size: %"PRIu32" bytes\n", lfs_cfg->block_size);
    printk("Prog size: %"PRIu32" bytes\n\n", lfs_cfg->prog_size);
}

static void user_button_interrupt_handler(void *handler_arg, cyhal_gpio_event_t event)
{
    ARG_UNUSED(handler_arg);
    ARG_UNUSED(event);

    /* notify thread that data is available */
```

### Creating your first Zephyr application for PSoC™ 6

```
    k_sem_give(&btn_isr_sema);
}

static void increment_boot_count(lfs_t *lfs, struct lfs_config *lfs_cfg)
{
    uint32_t boot_count = 0;
    lfs_file_t file;

    /* Mount the filesystem */
    int err = lfs_mount(lfs, lfs_cfg);

    /* Reformat if we cannot mount the filesystem.
     * This should only happen when littlefs is set up on the storage device for
     * the first time.
     */
    if (err) {
        printk("\nError in mounting. This could be the first time littlefs is used on the
storage device.\n");
        printk("Formatting the block device...\n\n");

        lfs_format(lfs, lfs_cfg);
        lfs_mount(lfs, lfs_cfg);
    }

    /* Read the current boot count. */
    lfs_file_open(lfs, &file, "boot_count", LFS_O_RDWR | LFS_O_CREAT);
    lfs_file_read(lfs, &file, &boot_count, sizeof(boot_count));

    /* Update the boot count. */
    boot_count += 1;
    lfs_file_rewind(lfs, &file);
    lfs_file_write(lfs, &file, &boot_count, sizeof(boot_count));

    /* The storage is not updated until the file_sd is closed successfully. */
    lfs_file_close(lfs, &file);

    /* Release any resources we were using. */
    lfs_unmount(lfs);

    /* Print the boot count. */
    printk("boot_count: %"PRIu32"\n\n", boot_count);
}

static void littlefs_task(void *dummy1, void *dummy2, void *dummy3)
```

### Creating your first Zephyr application for PSoC™ 6

```
{
    cy_rslt_t result;
    lfs_t lfs;
    struct lfs_config lfs_cfg;

    /* Remove warning for unused parameter */
    ARG_UNUSED(dummy1);
    ARG_UNUSED(dummy2);
    ARG_UNUSED(dummy3);

    /* Step 1: Get the default configuration for the block device.
     * Step 2: Initialize the lfs_config structure to zero (not required if it
     *         is a global variable)
     * Step 3: Create the block device
     * Step 4: Print the block device parameters such as erase block size
     * Step 5: Perform file system operations to increment the boot count
     */

#if(STORAGE_DEVICE_SD_CARD)
    lfs_sd_bd_config_t sd_bd_cfg;

    printk("Incrementing the boot count on SD Card...\n\n");

    /* Get the default configuration for the SD card block device. */
    lfs_sd_bd_get_default_config(&sd_bd_cfg);

    /* Initialize the pointers in lfs_cfg to NULL. */
    memset(&lfs_cfg, 0, sizeof(lfs_cfg));

    /* Create the SD card block device. */
    result = lfs_sd_bd_create(&lfs_cfg, &sd_bd_cfg);
    check_status("Creating SD card block device failed", result);
#else
    lfs_spi_flash_bd_config_t spi_flash_bd_cfg;

    printk("\nIncrementing the boot count on SPI flash...\n\n");

    /* Get the default configuration for the SPI flash block device. */
    lfs_spi_flash_bd_get_default_config(&spi_flash_bd_cfg);

    /* Initialize the pointers in lfs_cfg to NULL. */
    memset(&lfs_cfg, 0, sizeof(lfs_cfg));

    /* Create the SPI flash block device. */
```

### Creating your first Zephyr application for PSoC™ 6

```
    result = lfs_spi_flash_bd_create(&lfs_cfg, &spi_flash_bd_cfg);
    check_status("Creating SPI flash block device failed", result);
#endif /* #if(STORAGE_DEVICE_SD_CARD) */

    print_block_device_parameters(&lfs_cfg);
    increment_boot_count(&lfs, &lfs_cfg);

    /* Enable the user button interrupt */
    cyhal_gpio_enable_event(CYBSP_USER_BTN, CYHAL_GPIO_IRQ_FALL,
USER_BUTTON_INTERRUPT_PRIORITY, true);

    printk("Press the user button to format the block device or press reset to increment the
boot count again\n\n");

    /* Wait until the user button press is notified through the interrupt */
    while (true)
    {
        if (k_sem_take(&btn_isr_sema, K_MSEC(DEBOUNCE_DELAY_MS)) != 0)
        {
            /* Debounce the button press. */
            k_msleep(DEBOUNCE_DELAY_MS);

            if(!cyhal_gpio_read(CYBSP_USER_BTN)) { break; }
        }
    }

    /* User button is pressed. Format the block device. */
    printk("Formatting the block device...\n");
    lfs_format(&lfs, &lfs_cfg);
    printk("Formatting completed...\n");

#if(STORAGE_DEVICE_SD_CARD)
    lfs_sd_bd_destroy(&lfs_cfg);
#else
    lfs_spi_flash_bd_destroy(&lfs_cfg);
#endif /* #if(STORAGE_DEVICE_SD_CARD) */

    printk("Press reset to continue...\n");
}

void main(void)
{
    cy_rslt_t result;

    /* Initialize the pins */
```

```
    init_cycfg_pins();

    /* Initialize the user button used for erasing the block device. */
    result = cyhal_gpio_init(CYBSP_USER_BTN, CYHAL_GPIO_DIR_INPUT, CYHAL_GPIO_DRIVE_PULLUP,
CYBSP_BTN_OFF);
    CY_ASSERT (result == CY_RSLT_SUCCESS);

    /* Configure & the user button interrupt */
    gpio_btn_callback_data.callback = user_button_interrupt_handler;
    cyhal_gpio_register_callback(CYBSP_USER_BTN, &gpio_btn_callback_data);

    /* Enable global interrupts */
    __enable_irq();

    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printk("\x1b[2J\x1b[;H");

    printk("************* "
           "Littlefs File System on SD Card and QSPI NOR Flash "
           "************* \n\n");

    /* Started the created threads */
    k_thread_start(littlefs_task_id);
}

/* [] END OF FILE */
```

### 3.2.5 Modify the CMakeLists.txt file

*Note: Based on the modules generated, there will be minor changes required in CMakeLists.txt file.*

Do the following to modify the “CMakeLists.txt” file.

1. Specify the minimum required version of CMake:

```
cmake_minimum_required(VERSION 3.20.0)
```

2. CMake requires the `cmake_minimum_required()` call. It is also invoked by the Zephyr package in the next line. CMake will error out if its version is older than either the version in your *CMakeLists.txt* or the version number in the Zephyr package.

3. Add pointers for finding packages for build:

```
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
```

4. The `find_package()` function pulls in the Zephyr build system, which creates a CMake target named “app”. To include sources to the build, add sources to this target. The Zephyr package defines “Zephyr-Kernel” as a CMake project and enables support for C, CXX, and ASM languages.

### Creating your first Zephyr application for PSoC™ 6

5. Define the CMake project:

```
project(littlefs_filesystem)
```

The `project()` function defines your application's CMake project. Call this function after `find_package()` to avoid interference with Zephyr's project (Zephyr-Kernel).

6. Set relative path variables for PDL, HAL drivers, mtb-littlefs, serial-flash, and abstraction-rtos libraries which are required for the littlefs filesystem application:

```
set(PDL_DRIVER_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/mtb-pdl-cat1/drivers/source)
set(HAL_SOURCE_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/mtb-hal-cat1/COMPONENT_PSOC6HAL/source)
set(MTB_LITTLEFS_LIB_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../lib/mtb-littlefs)
set(SERIAL_FLASH_LIB_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../lib/serial-flash)
set(ABSTRACTION_RTOS_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/abstraction-rtos)
set(COMPONENT_ZEPHYR_DIR
${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../../zephyr/modules/hal_infineon/abstraction-rtos)
```

*Note:* *${ZEPHYR_HAL_INFINEON_MODULE_DIR} refers to C:\zephyrproject\modules\hal\infineon.*

7. Set a relative path variable for the littlefs library:

```
set(LITTLEFS_LIB_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../fs/littlefs)
set(LITTLEFS_BD_LIB_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../fs/littlefs/bd)
```

8. Include the library path of littlefs, mtb-littlefs, serial-flash libraries, and abstraction-rtos libraries:

```
zephyr_include_directories(${LITTLEFS_LIB_DIR})
zephyr_include_directories(${LITTLEFS_BD_LIB_DIR})
zephyr_include_directories(${MTB_LITTLEFS_LIB_DIR}/include)
zephyr_include_directories(${SERIAL_FLASH_LIB_DIR})
zephyr_include_directories(${ABSTRACTION_RTOS_DIR}/include)
zephyr_include_directories(${COMPONENT_ZEPHYR_DIR}/include/COMPONENT_ZEPHYR)
```

9. Find all the driver source files of littlefs, serial-flash, mtb-littlefs libraries, and abstraction-rtos:

```
FILE(GLOB littlefs_sources ${LITTLEFS_LIB_DIR}/*.c ${LITTLEFS_BD_LIB_DIR}/*.c)
FILE(GLOB serial_flash_sources ${SERIAL_FLASH_LIB_DIR}/*.c)
FILE(GLOB mtb_littlefs_sources ${MTB_LITTLEFS_LIB_DIR}/source/*.c)
FILE(GLOB abstraction_rtos_sources ${ABSTRACTION_RTOS_DIR}/source/*.c)
FILE(GLOB component_zephyr_sources ${COMPONENT_ZEPHYR_DIR}/source/COMPONENT_ZEPHYR/*.c)
```

10. Include the source files of littlefs, serial-flash, mtb-littlefs libraries and abstraction-rtos into the build system:

```
zephyr_library_sources(${littlefs_sources} ${serial_flash_sources} ${mtb_littlefs_sources}
${abstraction_rtos_sources} ${component_zephyr_sources})
```

11. Add definitions to the compiler command line:

### Creating your first Zephyr application for PSoC™ 6

```
add_definitions( -DCOMPONENT_CUSTOM_DESIGN_MODUS )
add_definitions( -DCY_RETARGET_IO_CONVERT_LF_TO_CRLF )
add_definitions( -DLFS_THREADSAFE )
```

12. Include the path for *src* and *GenaratedSource* subdirectories:

```
zephyr_include_directories(GeneratedSource src)
```

13. Find all the source files of the application:

```
FILE(GLOB app_sources GeneratedSource/*c src/*.c)
```

14. Add application source files to the 'app' target library, each on its own line:

```
target_sources(app PRIVATE ${app_sources})
```

See Application CMakeLists.txt to know more about this file.

The content of the CMakeLists.txt file should look as per the following snippet:

```
# SPDX-License-Identifier: Apache-2.0
cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(littlefs_filesystem)

set(PDL_DRIVER_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/mtb-pdl-cat1/drivers/source)
set(HAL_SOURCE_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/mtb-hal-cat1/COMPONENT_PSOC6HAL/source)
set(LITTLEFS_LIB_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../fs/littlefs)
set(LITTLEFS_BD_LIB_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../fs/littlefs/bd)
set(MTB_LITTLEFS_LIB_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../lib/mtb-littlefs)
set(SERIAL_FLASH_LIB_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../lib/serial-flash)
set(ABSTRACTION_RTOS_DIR ${ZEPHYR_HAL_INFINEON_MODULE_DIR}/abstraction-rtos)
set(COMPONENT_ZEPHYR_DIR
${ZEPHYR_HAL_INFINEON_MODULE_DIR}/../../../zephyr/modules/hal_infineon/abstraction-rtos)

zephyr_library_sources(${PDL_DRIVER_DIR}/cy_smif.c ${PDL_DRIVER_DIR}/cy_smif_memslot.c
${PDL_DRIVER_DIR}/cy_dma.c ${PDL_DRIVER_DIR}/cy_sd_host.c)
zephyr_library_sources(${HAL_SOURCE_DIR}/cyhal_qspi.c ${HAL_SOURCE_DIR}/cyhal_sdhc.c)

zephyr_include_directories(${LITTLEFS_LIB_DIR})
zephyr_include_directories(${LITTLEFS_BD_LIB_DIR})
zephyr_include_directories(${MTB_LITTLEFS_LIB_DIR}/include)
zephyr_include_directories(${SERIAL_FLASH_LIB_DIR})
zephyr_include_directories(${ABSTRACTION_RTOS_DIR}/include)
zephyr_include_directories(${COMPONENT_ZEPHYR_DIR}/include/COMPONENT_ZEPHYR)

FILE(GLOB littlefs_sources ${LITTLEFS_LIB_DIR}/*.c ${LITTLEFS_BD_LIB_DIR}/*.c)
```

```
FILE(GLOB serial_flash_sources ${SERIAL_FLASH_LIB_DIR}/*.c)
FILE(GLOB mtb_littlefs_sources ${MTB_LITTLEFS_LIB_DIR}/source/*.c)
FILE(GLOB abstraction_rtos_sources ${ABSTRACTION_RTOS_DIR}/source/*.c)
FILE(GLOB component_zephyr_sources ${COMPONENT_ZEPHYR_DIR}/source/COMPONENT_ZEPHYR/*.c)

zephyr_library_sources(${littlefs_sources} ${serial_flash_sources} ${mtb_littlefs_sources}
${abstraction_rtos_sources} ${component_zephyr_sources})

add_definitions( -DCOMPONENT_CUSTOM_DESIGN_MODUS )
add_definitions( -DCY_RETARGET_IO_CONVERT_LF_TO_CRLF )
add_definitions( -DLFS_THREADSAFE )

zephyr_include_directories(GeneratedSource src)
FILE(GLOB app_sources GeneratedSource/*c src/*.c)
target_sources(app PRIVATE ${app_sources})
```

### 3.2.6 Create Kconfig fragment

- Create at least one Kconfig fragment for your application (usually named “prj.conf”) and set the Kconfig option values required by your application there. See Kconfig Configuration. If no Kconfig options need to be set, create an empty file.

  Enable the Newlib variant in the *prj.conf* file as follows:

```
CONFIG_NEWLIB_LIBC=y
```

### 3.2.7 Create DeviceTree overlays

- Configure any DeviceTree overlays required by your application usually in a file named “app.overlay”. See DeviceTree overlays for more information. This is not required for the littlefs filesystem application.

### 3.2.8 Build the application and program

1. In the command prompt, navigate to the Zephyr application directory:

```
cd "C:\zephyrproject\zephyr_example_filesystem_littlefs"
```

2. Build the application by executing the following command:

```
west build -p auto -b cy8cproto_062_4343w
```

*Note:* *If the application throws any errors after building related to PDL/HAL, add the path of missing PDL/HAL source files with the source file name in the CMakeLists.txt file and build again.*

For example:

```
zephyr_library_sources(${PDL_DRIVER_DIR}/cy_smif.c)
zephyr_library_sources(${HAL_SOURCE_DIR}/cyhal_qspi.c)
```

3. After the application is built successfully, connect the board to your PC using the provided USB cable through the KitProg3 USB connector.
4. Open a terminal and select the KitProg3 COM port. Set the serial port parameters to 8N1 and 115200 baud.

### Creating your first Zephyr application for PSoC™ 6

5. Program the application into the board using the following command:

```
west flash
```

   Alternatively, you can use the CYPRESS™ Programmer to flash the application.

6. After programming, the application starts automatically. Confirm that "Littlefs File System on SD Card and QSPI NOR Flash" is displayed on the UART terminal.

   When the application is run for the first time, mounting the file system might result in error; if an error occurs, the memory is formatted and then the file system is mounted.
7. Press the reset button on the kit and observe that **boot_count** is incremented.
8. Press the user button on the kit to format the memory and reset boot_count. Press the reset button to increment boot_count again.
9. By default, the application uses the QSPI NOR flash as the storage device. To change the storage device to the SD card, set the value of the `STORAGE_DEVICE_SD_CARD` macro in *main.c* to '1', build the application, and program the kit.

See the README.md file of the littlefs filesystem application for more information.

# 4 Set up a debugger in Eclipse IDE for ModusToolbox™

## 4.1 Generate the project description file

1. In the command prompt, navigate to the Zephyr application directory:

```
cd "C:\zephyrproject\zephyr_example_filesystem_littlefs"
```

2. Build the application by executing the following command:

```
west build -b cy8cproto_062_4343w -- -G"Eclipse CDT4 - Ninja"
```

*Note: A different CMake generator is specified by the `-G"Eclipse CDT4 - Ninja"` argument. This will generate an Eclipse project description file ( .project) in addition to the usual Ninja build files.*

3. In Eclipse IDE for ModusToolbox™ 2.4, choose a new workspace.
4. Select **File** > **Import** and then select **General** > **Existing Projects into Workspace** to import your generated project.
5. Click **Select root directory** and navigate to your application *build* directory where the *.project* file is generated. Select your project from the list of projects and click **Finish**.
6. Wait for the project to load and ensure that it is successful.

## 4.2 Create a debugger configuration in Eclipse IDE for ModusToolbox™

1. Select **Run** > **Debug Configurations**.
2. Select **GDB OpenOCD Debugging**, click **New**
3. Configure the following options on the **Main** tab:

- Name: `littlefs_filesystem_debug`
- Project: Browse -> `littlefs_filesystem@build`
- C/C++ Application: <your Zephyr project repo> -> `zephyr/zephyr.elf`

### Set up a debugger in Eclipse IDE for ModusToolbox™

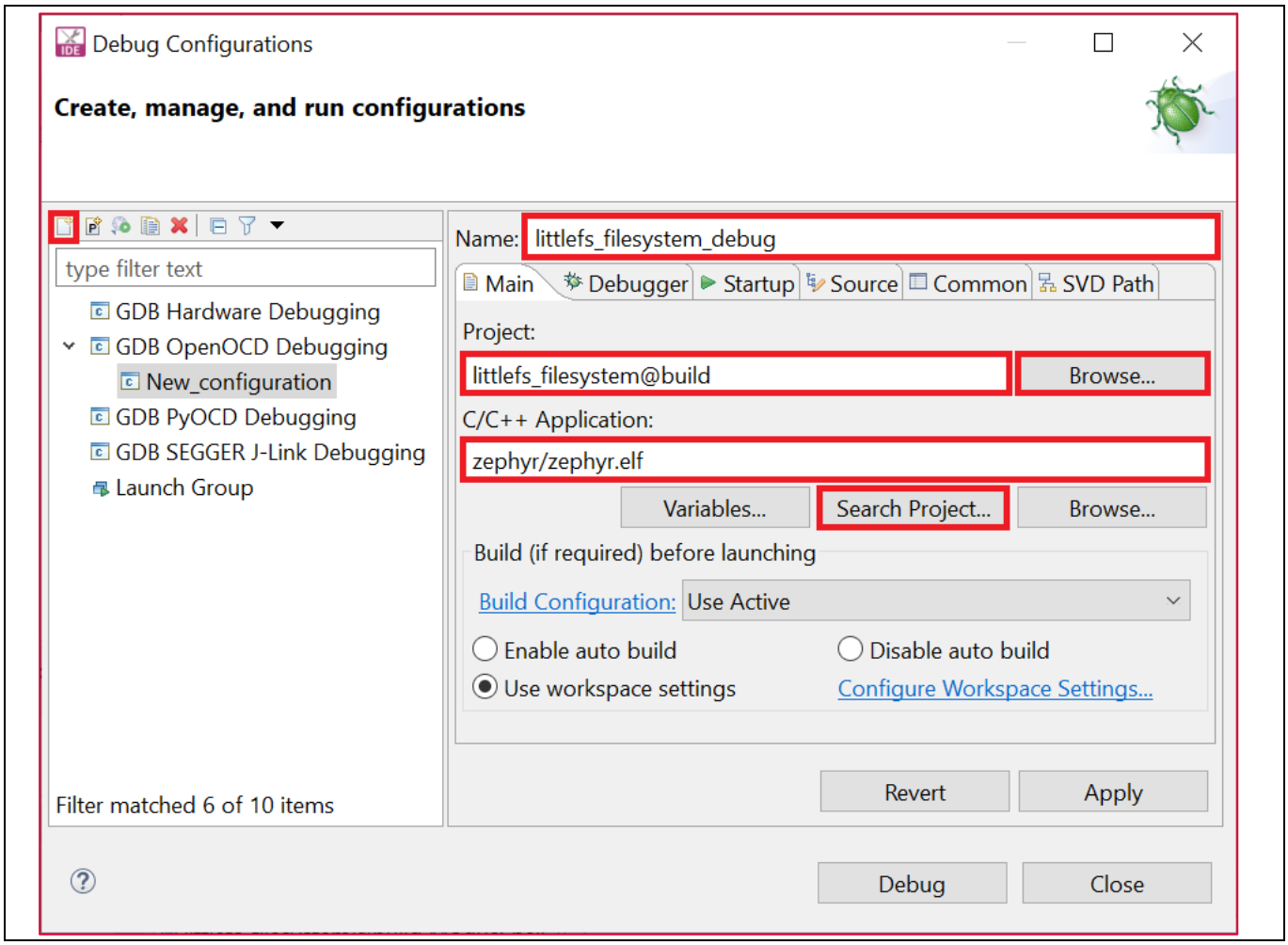


**Figure 2 Debugger settings for Main tab**

4. Navigate to the **Debugger** tab, and make the following changes:

a) OpenOCD Setup:

- Executable path: {$openocd_path}/{$openocd_executable}
- Config options:

```
-s "${openocd_path}/../scripts"
-c "source [find interface/kitprog3.cfg]"
-c "transport select swd"
-c "puts stderr {Started by GNU MCU Eclipse}"
-c "source [find target/psoc6_2m.cfg]"
-c "psoc6.cpu.cm4 configure -rtos auto -rtos-wipe-on-reset-halt 1"
-c "gdb_port 3332"
-c "psoc6 sflash_restrictions 1"
-c "init; reset init"
```

b) GDB Client Setup:

- Executable name: ${cy_tools_path:CY_TOOL_arm-none-eabi-gdb_EXE}

### Set up a debugger in Eclipse IDE for ModusToolbox™

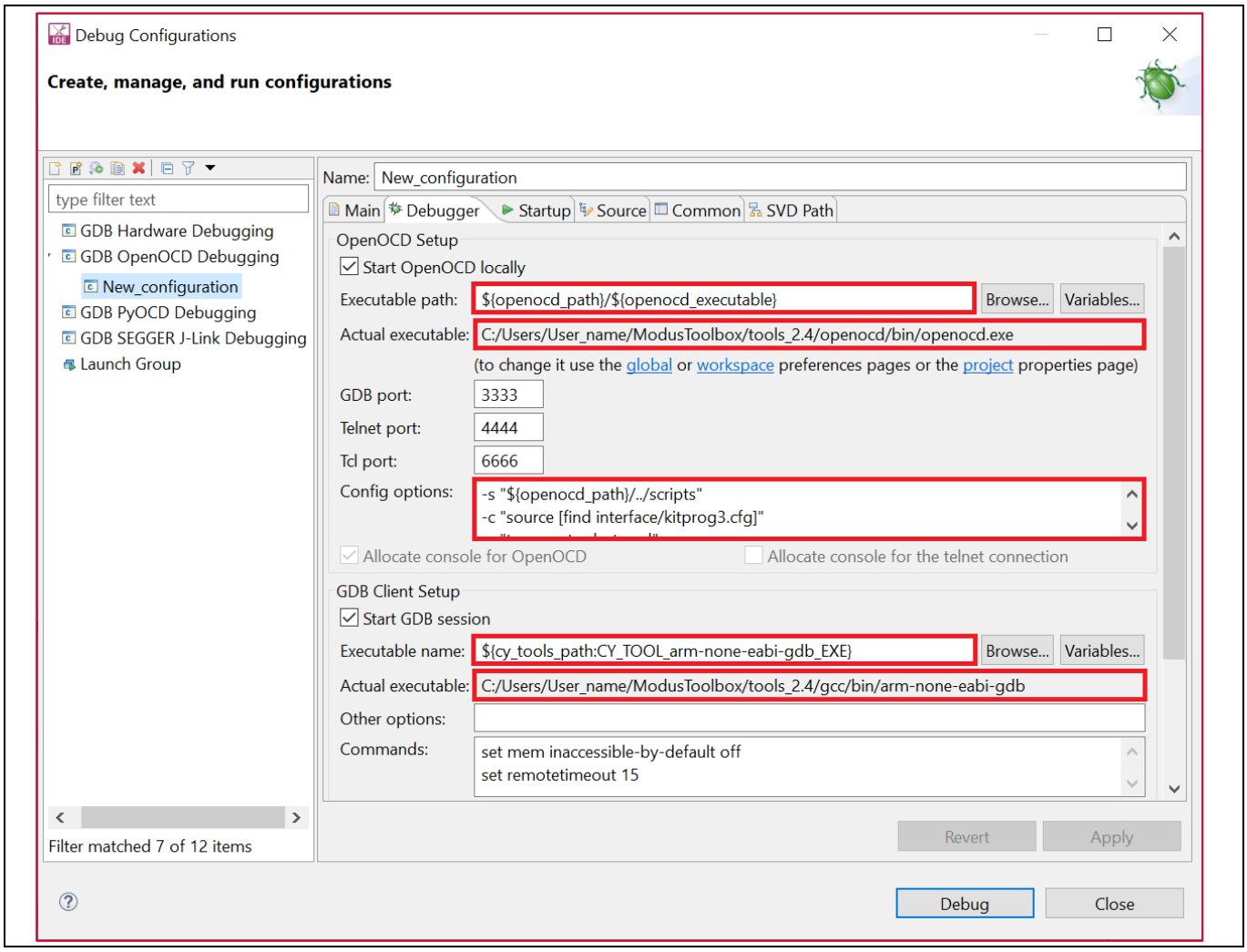


**Figure 3 Debugger settings for Debugger tab**

5. Navigate to the **Startup** tab, and make the following changes:

- Uncheck Initial Reset under Initialization Commands.
- Do the following in the **Run/Restart Commands** section:

a) Change the **Type** to **run.**

b) Add the following commands inside the textbox:

```
mon psoc6 reset_halt sysresetreq
flushregs
mon gdb_sync
stepi
```

### Set up a debugger in Eclipse IDE for ModusToolbox™

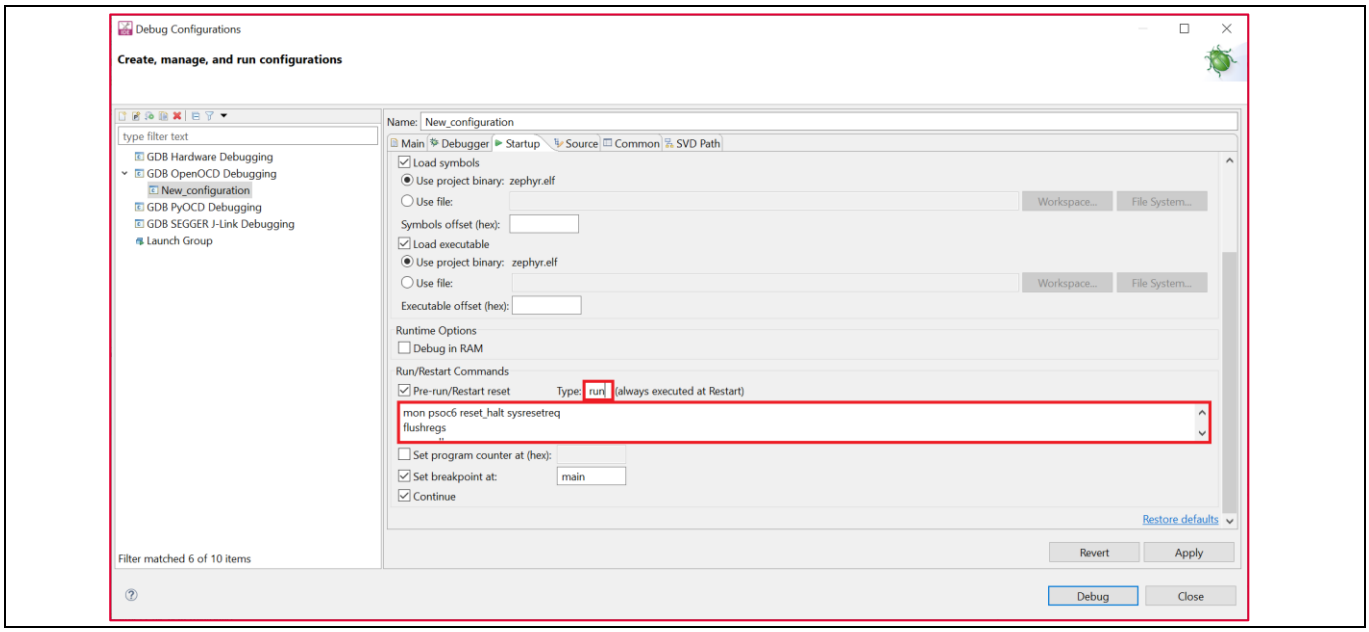


**Figure 4 Debugger settings for Startup tab**

6. Navigate to the **SVD Path** tab, and make the following changes:

- Set **File path** to: C:\zephyrproject\modules\hal\infineon\mtb-pdl-cat1\devices\COMPONENT_CAT1A\svd\psoc6_02.svd

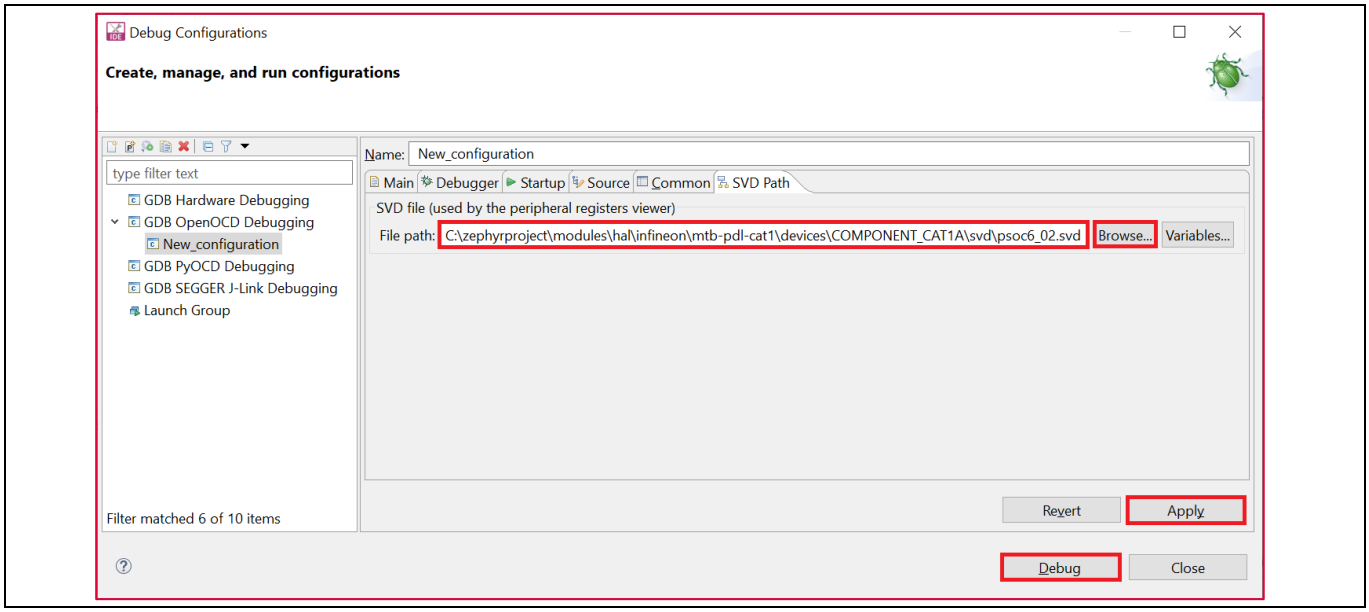


**Figure 5 Debugger settings for SVD Path tab**

7. Click **Apply** and then click **Debug**.

For more information on debugging, see the following:

- ModusToolbox™ user guide
- Zephyr documentation: Set up the debugging environment for Eclipse

# References

[1] AN228571 – Getting started with PSoC™ 6 MCU on ModusToolbox™ software

[2] Eclipse IDE for ModusToolbox™ user guide

[3] Getting started guide for Zephyr

[4] Code examples using ModusToolbox™ on GitHub

[5] mtb-pdl-cat1 – PSoC™ 6 Peripheral Driver Library (PDL)

[6] mtb-hal-cat1 – Hardware Abstraction Layer (HAL) library

[7] PSoC™ 6 MCU datasheets

[8] PSoC™ 6 MCU technical reference manuals

## Revision history

| Document revision | Date | Description of changes |
|---|---|---|
| ** | 2023-03-29 | Initial release. |
| *A | 2023-04-13 | Updated images. |

**Trademarks**
All referenced product or service names and trademarks are the property of their respective owners.

**Edition 2023-04-13**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**



**Do you have a question about this document?**

**Email:** erratum@infineon.com

**Document reference**

**002-36472 Rev. *A**

**Important notice**

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

**Warnings**

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.